# The Role of Latin Square in Cipher Systems: A Matrix Approach to Model Encryption Modes of Operation

Jiejun Kong
Computer Science Department
University of California, Los Angeles
jkong@cs.ucla.edu

**Abstract**

This paper studies the theoretic background of cryptographic modes of operation, in particular those modes proposed to ensure message privacy. A novel algebraic model is presented as an archetype of encryption design. In the ideal case, encrypting multiple messages is treated as inductively applying the algebraic operation ★, an operation corresponding to block-by-block processing, on Latin Squares over a sequence of finite groups $\{\mathbb{Z}_{r^n}, \mathbb{Z}_{r^{2*n}}, \mathbb{Z}_{r^{3*n}}, \cdots, \}$. We further show that a Latin Square cipher is a newly discovered hard-core function for any strong one-way length-preserving function. Based on the discovery, we propose a thesis that encryption modes of operation should implement cryptographically strong pseudorandom generators in the ideal case, so that the random oracle model can be used to justify the practice of replacing Latin Square ciphers with "good" implementations (e.g., AES).

Finally we present a cryptanalysis of NIST's standard modes of operation based on this work. The algebraic model shows that, even when an ideally strong one-way function is used, none of NIST's standard modes of operation (OFB, CFB, CTR, CBC) can produce cryptographically strong pseudorandom ensembles based on the ideal one-way function—the distinction of this work is to use formal method (rather than empirical attacks) to illustrate the design flaws in the standard modes of operation. As numerous security protocols are using the flawed modes of operation, we argue that these national standards should be repaired, and efficient repairs (double encryption) can be easily achieved.

## I. Introduction

A fundamental problem in cryptography is using a single secret key to process multiple messages in a cipher system with finite domain and range. Assuming a reasonable key lifetime, applications like data encryption, cryptographically strong hashing, timestamped multiple data signing demand a cipher system to maintain its security strength even when the same secret key is used many times. In the real world, cryptographic modes of operation have been widely used in practice to ensure privacy and integrity by reapplying the same secret key multiple times. This paper focuses on privacy oriented modes of operation (e.g., CBC, CFB, OFB modes, but not CBC-MAC, OCB modes etc.) and present an algebraic analysis of common modes of operation in use. Though some literatures [5][1] have addressed the same problem by both empirical and theoretic analysis, none of them has followed the same algebraic approach used in this work.

In our algebraic notations, realizing an ideal random oracle is equivalent to implementing a "good" one-way function over the finite group $\mathbb{Z}_{r^n}$. Then we define a novel operation "nominal construction ★" that inductively changes the base finite group to a sequence of finite groups $\{\mathbb{Z}_{r^n}, \mathbb{Z}_{r^{2*n}}, \mathbb{Z}_{r^{3*n}}, \cdots, \}$. The operation ★ is a well-defined algebraic operation on square matrices, and the induction is rendered on Latin Squares, which are specific kind of square matrices corresponding to the concept of *perfect system* (or a collection of trapdoor permutations with uniformly distributed key) [20].

By the algebraic model, using the ideal Latin Square random oracles to encrypt multiple blocks of data is equivalent to performing a sequence of nominal constructions over the sequence of finite groups. We show that Latin Squares with invariance properties can be used to construct a cryptographically strong pseudorandom generator (CSPRG). The construction itself, though studied under ideal conditions in this paper, can be applied under random oracle model[2] to justify encryption modes of operation design in the real world.

The applications of this algebraic model is demonstrated by related cryptanalysis against national standard modes widely used in the real world. Even when an ideally strong one-way function is used, none of NIST's standard

modes of operation (OFB, CFB, CTR, CBC) can produce cryptographically strong pseudorandom ensembles based on the ideal one-way function. As numerous security protocols are using the flawed modes of operation, we argue that these national standards should be repaired, and we show at least two efficient repairs can be easily achieved.

The remaining of the paper is organized as follows: Section II explains the notations used in this paper, in particular the algebraic encryption model, perfect system, and Latin Square ciphers. Then Section III introduces a new operation "nominal construction" $\bigstar$ and analyzes its algebraic properties. Based on the new notions, Section IV defines the concept of pseudoperfect system. In Section V and VI we show how to construct cryptographically strong pseudorandom generators from Latin Square ciphers. In Section VII we discuss the design flaws of NIST's standard modes of operation and the potential repairs. And finally Section VIII concludes the paper.

## II. NOTATIONS

### A. Common notations

Let $\mathbb{S}_q$ denote a finite set of size $q$. A function $f : \mathbb{S}_q \mapsto \mathbb{S}_q$ maps an element in $\mathbb{S}_q$ to another element. Let $\mathbb{F}_q$ be the set of all $q^q$ functions mapping $\mathbb{S}_q$ into $\mathbb{S}_q$. Let $\mathbb{P}_q \subset \mathbb{F}_q$ be the set of $q!$ such functions that are permutations. Let $\in_U$ denote selecting an element from a set following uniform distribution, for example, $x \in_U \mathbb{S}_q$.

We operate on specific finite sets known as "strings". $\mathbb{S}_{2^n} = \mathbb{Z}_2^n$ denotes the set of all $2^n$ binary strings of length $n$, and $\mathbb{Z}_2^+$ denotes binary strings of any length (the symbol $+$ denotes integer addition elsewhere, and the symbol $*$ or $\cdot$ denotes integer multiplication except in the notion of multiplicative group $\mathbb{Z}_n^*$). Similarly, let $r$ be any positive integer greater than 1. $\mathbb{S}_{r^n} = \mathbb{Z}_r^n$ denotes the set of all $r^n$ $r$-ary strings of length $n$ (i.e., $r$ is the radix), and $\mathbb{Z}_r^+$ denotes $r$-ary strings of any length.

An algebraic representation of $\mathbb{Z}_r^n$ is $\mathbb{Z}_{r^n}$, i.e., the set $\{0, 1, \cdots, r^n - 1\}$ as we can treat $r$ as radix and strings as numbers in position system. Hence the string comparison operator is defined as the integer comparison operator $\leq$ in $r$-ary position system[1].

An (endomorphic) encryption/encipher function is denoted as

$$T : \mathbb{S}_q \times \mathbb{S}_q \mapsto \mathbb{S}_q.$$

That is, by using a key $k \in \mathbb{S}_q$, a plaintext $m \in \mathbb{S}_q$ is encrypted into ciphertext $e = T(k, m) \in \mathbb{S}_q$. Due to Curry's work [6], for any function $f$ defined on a tuple type $D_1 \times D_2$ and with return type $R$,

$$f : (D_1 \times D_2) \mapsto R$$

there is a high-order function $F$ defined on the first domain that returns a function defined on the second domain and with return type $R$:

$$F : D_1 \mapsto (D_2 \mapsto R).$$

The second function $F$ is called the *curried* version of the first function $f$, and $f$ is called the *uncurried* version of $F$[2]. Later we will write the function returned by $F$ as $f_{D_1}$.

We obtain the curried result of the encipher $T$. Intuitively, the first argument (i.e, the key) is written as a subscript, $T_k$. In this notation, $T_k : \mathbb{S}_q \mapsto \mathbb{S}_q$ may be thought of as a set of $q$ functions indexed by key and is a subset of $\mathbb{F}_q$. Similarly, if we define its counterpart function $T'(m, k) = T(k, m)$ and obtain the curried result of the counterpart $T'_m : \mathbb{S}_q \mapsto \mathbb{S}_q$, the result is a set of $q$ functions indexed by plaintext.

The matrix representation of an encryption function $T : \mathbb{S}_q \times \mathbb{S}_q \mapsto \mathbb{S}_q$ is a square matrix[3] $L^T$ over $\mathbb{S}_q$:

$$\begin{array}{cc} & \begin{array}{cccccc} & m = & m_1 & m_2 & \cdots & m_q \end{array} \\ \begin{array}{c} k = \\ \\ \\ \\ \end{array} & \begin{array}{c} k_1 \\ k_2 \\ \vdots \\ k_q \end{array} \left[\begin{array}{cccc} e = & e_{k_1,m_1} & e_{k_1,m_2} & \cdots & e_{k_1,m_q} \\ & e_{k_2,m_1} & e_{k_2,m_2} & \cdots & e_{k_2,m_q} \\ & \vdots & \vdots & \ddots & \vdots \\ & e_{k_q,m_1} & e_{k_q,m_2} & \cdots & e_{k_q,m_q} \end{array}\right] \end{array}.$$

[1] Another algebraic representation of $(a_{n-1} \cdots a_1 a_0) \in \mathbb{Z}_r^n$ is a polynomial $f(r) = a_{n-1} \cdot r^{n-1} + \cdots + a_1 \cdot r + a_0$ where $a_i$ is from the commutative ring $Z_r$. $\mathbb{Z}_r^n$ is a special polynomial ring $Z_r[x]$ with $x = r$.

[2] In this paper we introduce Curry's work to avoid using the cumbersome notation of "collection of functions", which is actually a function uncurried on the collection index.

[3] Later in the paper we will use a cipher system $T$ and the corresponding square matrix $L^T$ as synonyms.

In a matrix $L$, let $L_{x,y}$ denote the element at row by $x$ and column indexed by $y$. In the matrix, distinct keys $k_i \in \mathbb{S}_q$ constitute the row indices, distinct plaintexts $m_j \in \mathbb{S}_q$ constitute the column indices, and every ciphertext $e_{k_i,m_j} = L^T_{k_i,m_j} = T(k_i, m_j) \in \mathbb{S}_q$. Note that the base set $\mathbb{S}_q$ of a cipher system sufficiently determines the dimensions of the corresponding square matrix. Thus we do not explicitly specify the dimensions of a square matrix as in other literatures.

The decryption/decipher function is denoted as $T^{-1}$ with identical domain and range as $T$. For any $k$, there is a $k^{-1}$, such that $T^{-1}_{k^{-1}} \circ T_k = I$ where $\circ$ denotes function composition and $I$ is the identity transformation $I(x) = x$. The general model does not require $k = k^{-1}$ or $T = T^{-1}$:

- For public key schemes, there exists a polynomial-time algorithm that can obtain the encryption/verification/public key $k$ from the decryption/signing/private key $k^{-1}$. But for sufficiently large $q$, there exists no polynomial-time algorithm that can obtain the private key $k^{-1}$ from the public key $k$ (i.e., it is one-way).
- For symmetric key schemes, $T = T^{-1}$, and there is a polynomial-time algorithm that can obtain $k$ and $k^{-1}$ from each other.

### *B. Perfect System and Latin Square*

Shannon [20] developed a mathematical theory for cryptography based on information theory. For random variables $K, M, E$ mapping into spaces $\mathcal{K} = \mathcal{M} = \mathcal{E} = \mathbb{S}_q$, respectively, the entropy difference $A(M, E) = H(M) - H(M|E)$ is the amount of information about $M$ which the adversary obtains. A perfect system is one in which $A(M, E)$ is zero, i.e., $H(M) = H(M|E)$. That is, if the secret key $k$ is uniformly chosen from the key space $\mathcal{K}$, then an adversary with any ciphertext has no choice but to select the pre-image plaintext following uniform distribution. Shannon also proved that $H(E) = H(E|M)$ is a necessary and sufficient condition for $A(M, E) = 0$. More formally, we define perfect system using random variables:

*Definition 1:* **(Perfect System):** Given a pair of functions $T : \mathbb{S}_q \times \mathbb{S}_q \mapsto \mathbb{S}_q$ and $T^{-1} : \mathbb{S}_q \times \mathbb{S}_q \mapsto \mathbb{S}_q$, the system $\langle T, T^{-1} \rangle$ is a *perfect system* if the following conditions hold:

1) *Identity transformation*: For any $k \in \mathbb{S}_q$, there exists $k^{-1} \in \mathbb{S}_q$, $k^{-1}$ can be computed in polynomial time from $k$. For any $m \in \mathbb{S}_q$, $T^{-1}(k^{-1}, (T(k, m))) = m$.
2) *Transitivity of uniform distribution*: Let $U_q$ be a random variables following uniform distribution over $\mathbb{S}_q$. Then both $T(U_q, m)$ and $T(m, U_q)$ are random variables following uniform distribution over $\mathbb{S}_q$. Similarly, both $T^{-1}(U_q, m)$ and $T^{-1}(m, U_q)$ are random variables following uniform distribution over $\mathbb{S}_q$
3) *Uniformly distributed key*: $k \in_U \mathbb{S}_q$. That is, the key $k$ is truly random. □

An important property of perfect system is "transitivity of uniform distribution", that is, given an arbitrary plaintext $m$, a perfect system will uniformly map it into any possible ciphertext because of the uniformly selected key. The correspondence between Latin Square and perfect system was also shown in the same reference [20]. In this paper a perfect system is treated as a special Latin Square cipher with truly random keys. In other words, a *Latin Square cipher* is a sub-perfect system operating on possibly non-truly random keys.

*Definition 2:* **(Latin Square):** A *Latin Square* over $\mathbb{S}_q$ is a $q \times q$ matrix $L$ over $\mathbb{S}_q$ whose entries are taken from $\mathbb{S}_q$ and which has the property that each symbol from $\mathbb{S}_q$ occurs exactly once in each row and exactly once in each column of $L$. In formal notions, (1) $L_{i_1,j} = L_{i_2,j}$ if and only if $i_1 = i_2$; (2) $L_{i,j_1} = L_{i,j_2}$ if and only if $j_1 = j_2$. □

In addition, if the base set $\mathbb{S}_q$ of a square matrix is a totally ordered set (e.g., $\mathbb{S}_{r^n} = \mathbb{Z}^n_r$), then the matrix's row indices and column indices are totally ordered. If the square matrix is a Latin Square, it can be normalized/reduced.

*Definition 3:* **(Normalized Latin Square):** A *normalized* (or *reduced*) Latin Square over a totally ordered set $\mathbb{S}_q$ has both its first row and first column ordered by the element comparison operation $\leq$ of the set. □

*Example 1:* A normalized Latin Square over $\mathbb{S}_{2^2} = \mathbb{Z}^2_2$ is:

$$\begin{array}{cc} & m= \\ k= & \begin{array}{c}00\\01\\10\\11\end{array} \end{array} \begin{array}{c} \begin{array}{cccc}00&01&10&11\end{array} \\ \left[\begin{array}{cccc} e=00&01&10&11\\ 01&00&11&10\\ 10&11&00&01\\ 11&10&01&00 \end{array}\right] \end{array}. \quad \square$$

By the matrix representation of cipher systems, Latin Squares over $\mathbb{S}_q$ becomes a collection of permutations indexed by either the key $k$ or the plaintext $m$. $T_k$, the curried result of $T$, is the set of rows that are mappings between plaintext $m$ and ciphertext $e$. $T'_m$, the curried result of the counterpart $T'(m,k) = T(k,m)$, corresponds to the set of columns that are mappings between ciphertext $e$ and key $k$. It is important to point out that among $q!$ permutations in $\mathbb{P}_q$, there are only $q$ of them are qualified to be a row or a column in the Latin Square. They must pairwisely map distinct plaintexts (or keys) into distinct ciphertexts. It is easy to verify that the bitwise exclusive-OR operation "$\oplus$" is an implementation of the Latin Square $\left[\begin{array}{cc}0&1\\1&0\end{array}\right]$ over $\mathbb{S}_{2^1} = \mathbb{Z}_2^1$, and the negation of $\oplus$ is an implementation of the Latin Square $\left[\begin{array}{cc}1&0\\0&1\end{array}\right]$ over $\mathbb{Z}_2^1$.

## III. Nominal Construction

In this section we define a new operation $\bigstar$ on matrices and then study its algebraic properties. In the next section we will show the correspondence between this operation and empirical practice.

*Definition 4:* **(Nominal Construction):** Given a $x_a \times y_a$ matrix $A$ over $\mathbb{Z}_r^{n_a}$ and a $x_b \times y_b$ matrix $B$ over $\mathbb{Z}_r^{n_b}$, the *nominal construction* $\bigstar$ on $A$ with $B$ generates a $(x_a \cdot x_b) \times (y_a \cdot y_b)$ matrix $C = A \bigstar B$ over $\mathbb{Z}_r^{n_a+n_b}$:

1) ***Initialization***: $C$ has $x_b \cdot y_b$ sub-matrices of dimension $x_a \times y_a$. Let $C(i', j')$ denote the sub-matrix at $i'$-th row and $j'$-th column at the granularity of sub-matrix. Each of the sub-matrix is initialized as $A$. In other words, $C(i', j')_{i,j} = A_{i,j}$ and in shorthand $C(i', j') = A$.
2) ***Prefix***: Each element of a sub-matrix is prefixed with the corresponding element in $B$. In other words, each element in $C(i, j)$ is prefixed with $B$'s element $B_{i,j}$.

An algebraic representation of the entire procedure is $C(i', j')_{i,j} = r^{n_a} \cdot B_{i',j'} + A_{i,j}$. More precisely, let $x/y$ be the quotient of integer division and $x\%y$ be the remainder of integer division,

$$C_{i,j} = r^{n_a} \cdot B_{i/x_a, j/y_a} + A_{i\%x_a, j\%y_a}.$$

□

Note that each element in $A, B, C$ is a string comprised of $r$-ary integers. A matrix element of length $n$ is in the set $\mathbb{Z}_r^n \subset \mathbb{Z}_r^+$.

*Example 2:* Here is an example of nominal construction on a $2 \times 2$ matrix over $\mathbb{Z}_2^1$ with a $2 \times 3$ matrix over $\mathbb{Z}_2^1$. The result is a $4 \times 6$ matrix over $\mathbb{Z}_2^2$.

$$\left[\begin{array}{cc}0&1\\1&0\end{array}\right] \bigstar \left[\begin{array}{ccc}0&1&0\\1&0&1\end{array}\right] = \left[\begin{array}{ccc|ccc} 00&01&00&10&11&10\\ 01&00&01&11&10&11\\ \hline 10&11&10&00&01&00\\ 11&10&11&01&00&01 \end{array}\right]. \quad \square$$

*Theorem 5:* Given a Latin Square $A$ over $\mathbb{Z}_r^{n_a}$ and a Latin Square $B$ over $\mathbb{Z}_r^{n_b}$, the nominal construction $\bigstar$ on $A$ with $B$ generates a new Latin Square $C = A \bigstar B$ over $\mathbb{Z}_r^{n_a+n_b}$.

***Proof***: Any two elements at $C$'s same row will be different following either of the two cases: (1) They are in different sub-matrices, thus the prefix is different because $B$ is a Latin Square; (2) They are in the same sub-matrix, thus the initialized part is different because $A$ is a Latin Square.

The same argument also applies to any two elements at same column. Thus $C$ is a Latin Square.

Each element of $C$ has $n_a$ $r$-ary integers in its initialized part, and $n_b$ $r$-ary integers in its prefix part. It is a string of $r$-ary integers in length $n_a + n_b$. ∎

*Corollary 6:* Given a normalized Latin Square $A$ over $\mathbb{Z}_r^{n_a}$ and a normalized Latin Square $B$ over $\mathbb{Z}_r^{n_b}$, the nominal construction $\bigstar$ on $A$ with $B$ generates a new normalized Latin Square $C = A \bigstar B$ over $\mathbb{Z}_r^{n_a+n_b}$.

***Proof***: By Theorem 5, $C$ is a Latin Square over $\mathbb{Z}_r^{n_a+n_b}$.

The first row of $C$ is ordered because: (1) the most significant $n_b$ $r$-ary integers of the row elements are ordered because $B$ is a normalized Latin Square; (2) the least significant $n_a$ $r$-ary integers of the row elements are ordered because $A$ is a normalized Latin Square.

The same argument also applies to the first column. Thus $C$ is a normalized Latin Square. ∎

Given certain induction bases, nominal construction ★ can be used to construct algebraic structures.

*Definition 7:* **(Binary 1-power Nominal Latin Squares):** For all $x \geq 1$, a *binary 1-power nominal Latin Square* $L_{2,1}^{(x)}$ is over the set $\mathbb{Z}_2^x$ and constructed by mathematical induction in exactly $x$ rounds:

- The generator $L_{2,1}^{(1)}$ is either of the two Latin Squares over $\mathbb{Z}_2^1$, i.e., $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ or $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
- $L_{2,1}^{(x+1)}$ is constructed as the nominal construction on $L_{2,1}^{(x)}$ with $L_{2,1}^{(1)}$. That is,

$$L_{2,1}^{(x+1)} = L_{2,1}^{(x)} \bigstar L_{2,1}^{(1)}.$$

In particular, the *normalized* binary 1-power nominal Latin Square $L_{2,1}^{N(x)}$ is uniquely constructed from the normalized one. □

Any $L_{2,1}^{N(x)}$ is normalized by construction according to Corollary 6. For example, The simplest construction is $L_{2,1}^{N(2)} = \begin{bmatrix} 00 & 01 & 10 & 11 \\ 01 & 00 & 11 & 10 \\ 10 & 11 & 00 & 01 \\ 11 & 10 & 01 & 00 \end{bmatrix}$. Later in this paper we will show the relation between stream ciphers and $L_{2,1}^{N(x)}$.

It is easy to verify that ★ forms an Abelian semigroup on binary 1-power nominal Latin Squares.

*Theorem 8:* $\langle \mathbb{L}_{\mathbf{2,1}}, \bigstar \rangle$ forms an infinite Abelian semigroup on nominal Latin Squares $\mathbb{L}_{\mathbf{2,1}} = \bigcup_{i=1}^{\infty} L_{2,1}^{(i)}$.

***Proof***: Given any binary 1-power nominal Latin Square $A = L_{2,1}^{(n_a)}$, and any binary 1-power nominal Latin Square $B = L_{2,1}^{(n_b)}$. Their nominal construction $Y = A \bigstar B$ is in the set

$$Y = \underbrace{L_{2,1}^{(1)} \bigstar \cdots \bigstar L_{2,1}^{(1)}}_{n_a} \bigstar \underbrace{L_{2,1}^{(1)} \bigstar \cdots \bigstar L_{2,1}^{(1)}}_{n_b} = L_{2,1}^{(n_a+n_b)} \subset \mathbb{L}_{\mathbf{2,1}}.$$

$A \bigstar B = L_{2,1}^{(n_a+n_b)} = L_{2,1}^{(n_b+n_a)} = B \bigstar A$, so ★ is commutative.

For any binary 1-power nominal Latin Square $C = L_{2,1}^{(n_c)}$, $(A \bigstar B) \bigstar C = A \bigstar (B \bigstar C) = L_{2,1}^{(n_a+n_b+n_c)}$. Thus ★ is associative. ∎

More generally, if the generator $L_{2,1}^{(1)}$ of the semigroup $\mathbb{L}_{\mathbf{2,1}}$ is substituted with an arbitrary Latin Square $L_{r,n}^{(1)}$ over $\mathbb{Z}_r^n$, we have following definitions and theorems.

*Definition 9:* **($r$-ary $n$-power Nominal Latin Squares):** An *$r$-ary $n$-power nominal Latin Square* $L_{r,n}^{(x)}$ is over the set $\mathbb{Z}_r^{n \cdot x}$ and constructed by mathematical induction in exactly $x$ rounds:

- The generator $L_{r,n}^{(1)}$ is an arbitrary Latin Square over $\mathbb{Z}_r^n$.
- $L_{r,n}^{(x+1)}$ is constructed as the nominal construction on $L_{r,n}^{(x)}$ with $L_{r,n}^{(1)}$. That is, $L_{r,n}^{(x+1)} = L_{r,n}^{(x)} \bigstar L_{r,n}^{(1)}$. □

*Theorem 10:* $\langle \mathbb{L}_{\mathbf{r,n}}, \bigstar \rangle$ forms an infinite semigroup on Latin Squares $\mathbb{L}_{\mathbf{r,n}} = \bigcup_{i=1}^{\infty} L_{r,n}^{(i)}$.

***Proof***: The proof is identical to the proof of Theorem 8, except any occurrence of $2, 1$ is replaced by $r, n$ (which means the term "bit" is replaced by "$r$-ary integer" and bit-length is not always 1). ∎

Intuitively, the closure law of semigroup ensures that all members of this semigroup share system-level property with its generator. And the associativity law of semigroup ensures the order of operations is trivial. These laws are useful in constructing the pseudoperfect systems described below.

## IV. Pseudoperfect System

### A. Construction of Pseudoperfect System

We employ random oracle model [2] and assume the existence of an "ideal" implementation of Latin Square. The random oracle is named as *Latin Square Oracle* and will be replaced by actual implementations following the random oracle model.

*Definition 11:* **(Latin Square Oracle):** An *$r$-ary $n$-power Latin Square Oracle (LaSO)* is a random oracle that implements $L^{(1)}_{r,n}$, i.e., an arbitrary Latin Square over $\mathbb{Z}^n_r$. □

By the random oracle model we assume such a LaSO $L^T = L^{(1)}_{r,n}$ is realizable. Now we show that the empirical method of processing data block-by-block actually corresponds to nominal construction ★.

*Definition 12:* **(Little-endian System):** Let $B_0, B_1, \cdots, B_{x-1}$ denote a sequence of $x$ blocks of bits (i.e., strings from $\mathbb{Z}^+_2$), and $b_{i(0)}, b_{i(1)}, \cdots, b_{i(n-1)}$ denote the $n$ bits of block $B_i$. A *little-endian system* stores the blocks from left to right according to the order of block index, with the leftmost $B_0$ as the least significant block and rightmost $B_{x-1}$ as the most significant block. Within each block, the storage is implementation-defined, with $b_{i(n-1)}$ as the most significant bit and $b_{i(0)}$ as the least significant bit.

If we replace each bit with $r$-ary integer, we obtain a little-endian system over $\mathbb{Z}^+_r$. □

Such storage protocol is needed in computer systems to store multiple blocks of data and a block is normally called a "byte". For example, a hexadecimal bit sequence of multiple bytes "$0x12345678$" is stored as "$78\ 56\ 34\ 12$" in little-endian systems[4].

*Definition 13:* **(Pseudoperfect System):** Let $x$ be a polynomial. An *$r$-ary $n$-power nominal system of polynomial degree* is a cipher system that uses $r$-ary $n$-power LaSO to produce $n{\cdot}x$ long $r$-ary ciphertext from $n{\cdot}x$ long $r$-ary plaintext and $n{\cdot}x$ long $r$-ary key. The encryption is accomplished in exactly $x$ rounds: For all $1{\leq}i{\leq}x$, the ciphertext block $EB_i$ is produced by the LaSO $L^T$ from the plaintext block $MB_i$ and the key block $KB_i$.

An *$r$-ary $n$-power pseudoperfect system of polynomial degree* is a nominal system of the same metrics, and the $n{\cdot}x$ long $r$-ary key is generated from a $n$ long $r$-ary truly random key by a pseudorandom generator (PRG). If the PRG is cryptographically strong (i.e., CSPRG, so no Turing-complete algorithm can differentiate the pseudorandom result from truly random integers with non-negligible probability), then the system is called *$r$-ary $n$-power cryptographically strong pseudoperfect system of polynomial degree*. □

Figure 1 depicts an $r$-ary $n$-power pseudoperfect system. Note that it is not allowed to change the implementation of LaSO during the entire process, that is, the same LaSO is used to process all blocks. We prove that the "nominal system", the construction depicted inside the solid line in Figure 1, is a Latin Square.

*Theorem 14:* An $r$-ary $n$-power nominal system of degree $x$ constitutes a Latin Square cipher over $\mathbb{Z}^{n{\cdot}x}_r$.

***Proof***: We prove the theorem by induction: (1) If $x = 1$, $L^T$ is a perfect system, the theorem is true by assumption; (2) Suppose the theorem is true when $x = l$, we denote the corresponding perfect system and LaSO as $T'$ and $L^{T'}$, respectively. For $x = l + 1$, the procedure adds a new most significant $n$ bits to the plaintext, key, and ciphertext, respectively. Let $MB_l$ denote the new block in the plaintext, $KB_l$ denote the new block in the key, and $EB_l$ denote the new block in the ciphertext. Without loss of generality, the procedure is divided into two steps:

- We use the LaSO $L^T$ to produce $EB_l$ from $MB_l$ and $KB_l$. In little-endian systems, this step produces the most significant $n$ bits of the ciphertext from the most significant $n$ bits of the key and the plaintext. This is equivalent to identifying a sub-Latin-Square from the $r^n{\cdot}r^n$ choices that in turn were created in the prefixing step of nominal construction.
- We use the LaSO $L^{T'}$ to process the remaining least-significant $l$ blocks as usual. In other words, no change is made in processing the least significant $n{\cdot}l$ bits of ciphertext from the key and the plaintext. This is equivalent to applying the Latin Square created in the initialization step of nominal construction.

According to Theorem 10, the closure law of $\langle \mathbb{L}_{\mathbf{r},\mathbf{n}}, \bigstar \rangle$ ensures the result is exactly the Latin Square $L^{(x)}_{r,n}$. *Like a perfect system, a pseudoperfect system is also a Latin Square cipher. The difference between them is how the input key is chosen: (1) For a perfect system, the $n{\cdot}x$ long $r$-ary key is truly random; (2) For a pseudoperfect system, the input key is not truly random. But if the PRG generates a cryptographically strong pseudorandom ensemble, then the $n{\cdot}x$ long $r$-ary key is a pseudorandom sequence that cannot be differentiated from truly random sequence*

[4]The storage model is for the ease of presentation only. If we use "big-endian" systems (i.e., blocks are stored in reversed order), then the prefix operation in Definition 4 should be changed to postfix operation, and the comparison operator $\leq$ should be re-defined.

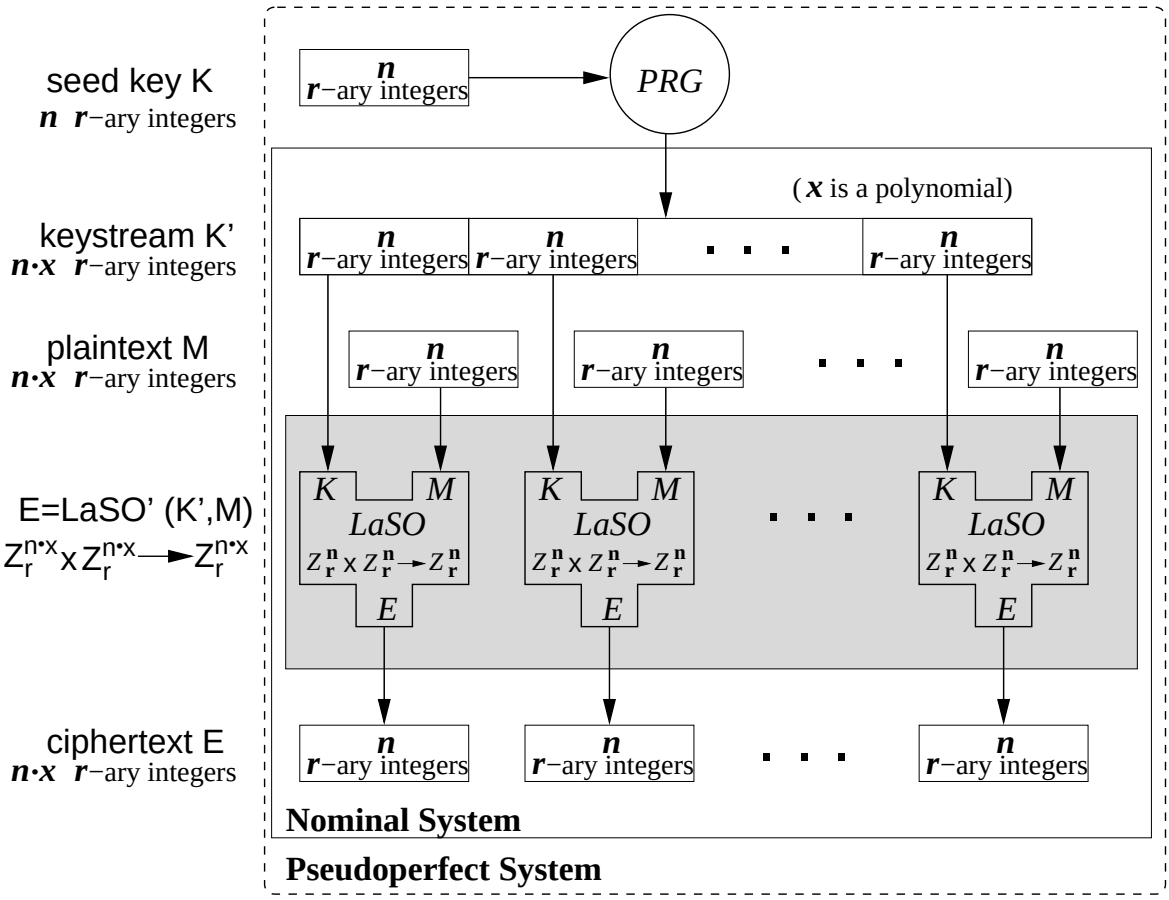


Fig. 1. **Construction of Pseudoperfect System**

*by any Turing-complete algorithm in polynomial time.* ∎

Cryptographically strong pseudoperfect system is the ideal case of encryption modes of operation. This algebraic view of modes of operation is more general than the classic notion of one-time pad (OTP). In a pseudoperfect system, $L^T$ is a Latin Square over $\mathbb{Z}_r^n$ for *any* $r$ and *any* $n$. The generator $L_{2,1}^{(1)}$ (implemented by bitwise exclusive-OR $\oplus$) used in OTP is merely a special case.

- In OTP, $r = 2$ and $n = 1$. Given the cipher $T$ and a known-plaintext $m$, $T_m$ must be one-way to prevent the adversary from discovering the key. However, there is no practical way to implement a one-way $L_{2,1}^{(1)}$, which is vulnerable to exhaustive search.
- In pseudoperfect system, $r$ and $n$ can be sufficiently but reasonably large. We will show below that a large $r$-ary $n$-power one-way LaSO can be realized and can be replaced by some "good" implementations of one-way function. For example, for DLP-based implementations, $r$ is a large strong prime and $n = 1$; for RSA and Rabin function, $r$ is the product of two large primes and $n = 1$; for AES, $r = 2$ and $n = 128$. Breaking the one-way property of these implementations equals to solving some computationally hard problems.

## V. Relation between LaSO and One Way Function

In this section we will firstly propose a slightly modified probabilistic polynomial time model based on well-studied concepts in foundations of cryptography. Then we will propose useful concepts needed to construct cryptographically strong pseudoperfect systems: (1) The composition of any Latin Square cipher and any one way function is yet another one way function; (2) Some Latin Square ciphers are one-way functions that can be used to realize pseudorandom generator.

### *A. Generalized probabilistic computation model and One-way Function (OWF)*

In our algebraic notations, realizing an ideal random oracle is equivalent to implementing a "good" one-way function over the finite group $\mathbb{Z}_r^n$ (i.e., $\mathbb{Z}_{r^n}$). The concept of one-way function is defined on polynomial relation between the input length and output length. Here we follow the common definition [11]:

*Definition 15:* **(One-way Function):** A function $f : \mathbb{Z}_2^+ \mapsto \mathbb{Z}_2^+$ is a (strong) *one-way* function if the following two conditions hold:

1) *Easy to compute*: There exists a deterministic polynomial-time algorithm $A$ such that on input $x$ it outputs $f(x)$, i.e., $A(x) = f(x)$.
2) *Hard to invert*: For every probabilistic polynomial-time algorithm $A'$, every positive polynomial $P(\cdot)$, and all sufficiently large $n$,
$$\mathsf{Pr}[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{P(n)}$$
where $U_n$ denotes a random variable uniformly distributed over $\mathbb{Z}_2^n$, and the auxiliary input $1^n$ gives the length of the desired output $n$ in unary notation. In particular, $\in$ can be replaced by $=$ if $f$ is bijective, and the auxiliary input $1^n$ is redundant if the one-way function is endomorphic $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^n$. □

The existence of one-way functions is not proven. Yet a number of conjectured one-way functions are routinely used in commerce and industry, such as Discrete Logarithm [8], RSA function [18], Rabin function [16], Feistel structures [10], and Substitution-Permutation Networks. Most existing cryptanalysis on their one-way property is based on binary system. If we replace 2 with arbitrary radix $r$, then we switch the study of one-way property from $\mathbb{Z}_2^+$ to $\mathbb{Z}_r^+$. This can be simply justified by changing the alphabet used in the corresponding probabilistic Turing Machines—we can replace the binary alphabet set $\mathbb{Z}_2 = \{0, 1\}$ with the $r$-ary alphabet set $\mathbb{Z}_r = \{0, 1, \ldots, r-1\}$, and the new probabilistic Turing Machine tosses an $r$-face dice rather than a 2-face coin. The binary probabilistic Turing Machine is a special case of this more general computation model with $r = 2$.

*Definition 16:* **($r$-ary Bounded-Probability Time, $\mathbb{BPP}$):** Let $M(x)$ be the random variable denoting the output of an $r$-ary probabilistic machine $M$. Let
$$\mathsf{Pr}[M(x) = y] = \frac{|\{d \in \mathbb{Z}_r^{t_M(x)} : \ M_d(x) = y\}|}{r^{t_M(x)}}$$
where $d$ is an $r$-face dice throw, $t_M(x)$ is the number of dice throws made by $M$ on input $x$, and $M_d(x)$ denotes the output of $M$ on input $x$ when $d$ is the outcome of its dice throws.

We say that $L$ is recognized by the $r$-ary probabilistic polynomial-time Turing Machine $M$ if

- for every $x \in L$ it holds that $\mathsf{Pr}[M \ accepts \ x] \geq \frac{1}{2} + \frac{1}{P(n)}$ for every polynomial $P(\cdot)$.
- for every $x \notin L$ it holds that $\mathsf{Pr}[M \ accepts \ x] \leq \frac{1}{2} - \frac{1}{P(n)}$ for every polynomial $P(\cdot)$.

$\mathbb{BPP}$ is the class of languages that can be recognized by an $r$-ary probabilistic polynomial time Turing Machine. □

It is clear that the complexity classes of $\mathbb{P}, \mathbb{NP}, \mathbb{PSPACE}, \mathbb{NPSPACE}$ are unchanged by choosing different alphabets in Turing Machines. Moreover, the following theorem justifies the conclusion that the probabilistic computation model $\mathbb{BPP}$ is also unchanged by radix conversion.

*Theorem 17:* Let random variable $X$ denote the distribution of $n_1$ long $r_1$-ary string $x = x_1, x_2, \ldots, x_{n_1}$. If $x_i \in_U \mathbb{Z}_{r_1}$, then $X$ can be reduced in polynomial time to a correspondence $Y$ in $r_2$-ary system such that the distribution $Y$ for $r_2$-ary string $y = y_1, y_2, \ldots, y_{n_2}$ satisfies $y_i \in_U \mathbb{Z}_{r_2}$.

***Proof***: Firstly, we can treat $X$'s sample space as $r_1^{n_1}$ "pigeon holes". The combinations of $x_i$'s fill the holes once, hence $X$ is a uniform distribution on $\mathbb{Z}_{r_1}^{n_1}$.

For any $x \in_U X$, in a little-endian system we have $x = x_{n_1} \cdot r_1^{n_1 - 1} + \ldots + x_2 \cdot r_1^1 + x_1$, then we change the representation to be $y = y_{n_2} \cdot r_2^{n_2 - 1} + \ldots + y_2 \cdot r_2^1 + y_1$. To obtain a truly random variable $Y$, we can apply the "padding argument" and pad some truly random bits to $x$. The padding is equivalent to turning the number of "pigeon holes" from $r_1^{n_1}$ to $r_2^{n_2}$. Then the radix conversion procedure can be accomplished in polynomial time by repetitively producing the remainder and quotient of $x/r_2$.

Finally, if $\exists y_i, y_i \notin_U \mathbb{Z}_{r_2}$ and $w$ is the value not in the distribution, then there are $\prod_{0 \le j < n_2, j \ne i} r_2 = r_2^{n_2-1}$ empty "pigeon holes" caused by $w$, hence $Y$ is not a uniform distribution over $\mathbb{Z}_{r_2}^{n_2}$. This contradiction proves $\forall y_i, y_i \in_U \mathbb{Z}_{r_2}$. ∎

An $r_1$-ary probabilistic Turing Machine can be viewed as having two input tapes: (1) a real $r_1$-ary input $x$ of length $|x|$ and (2) a uniformly chosen $d \in \mathbb{Z}_{r_1}^{t_M(x)}$ playing the role of a possible outcome for a sequence of dice throws. Then we can always convert the input tape $x$ to its $r_2$-ary equivalence with length $|x| \cdot \lceil \log_{r_2} r_1 \rceil$ and convert the sequence on dice-throw tape to its $r_2$-ary equivalence of length $t_M(x) \cdot \lceil \log_{r_2} r_1 \rceil$. Polynomial constraint on input length is unchanged as $P(t_M(x) \cdot \lceil \log_{r_2} r_1 \rceil)$ is always a polynomial if $P(t_M(x))$ is a polynomial of $|x|$.

In particular, we can always convert an $r$-ary random input into the corresponding binary representation, then run binary probabilistic Turing Machines to process the input, and if necessary convert the binary result back to the $r$-ary representation. Therefore, from this point on we will discuss one-way functions and pseudorandom generators in $r$-ary systems, where a single $r$-ary integer plays the role of a binary "bit".

### *B. Composition of OWF and LaSO*

The following theorem shows the relation between one-way functions and Latin Square ciphers.

*Theorem 18:* Function compositions of a bijective endomorphic one-way function $f : \mathbb{Z}_r^n \mapsto \mathbb{Z}_r^n$ and a LaSO $\{T, T^{-1}\}$ over $\mathbb{Z}_r^n$ are one-way functions. That is, $\{f \circ T_k, T_k \circ f, f \circ T_m, T_m \circ f\}$ and $\{f \circ T_{k^{-1}}^{-1}, T_{k^{-1}}^{-1} \circ f, f \circ T_e^{-1}, T_e^{-1} \circ f\}$ are sets of one-way functions from $\mathbb{Z}_r^n$ to $\mathbb{Z}_r^n$.

***Proof***: As any key assignment always returns permutation from a Latin Square cipher, we need to prove permutations over $\mathbb{Z}_r^n$ do not change the one-way property.

For $f \circ T_k$, the proof is divided into three steps:

1) As a permutation on $U_n$ is also a uniform distribution, $V = T_k(U_n)$ is a random variable uniformly distributed over $\mathbb{Z}_r^n$. Thus by one-way function's definition, for every probabilistic polynomial-time algorithm $A'$, every positive polynomial $P(\cdot)$, and all sufficiently large $n$
$$\mathsf{Pr}[A'(f(V)) = f^{-1}(f(V))] < \frac{1}{P(n)}.$$
2) Based on step 1, we need to prove for all sufficiently large $n$
$$\mathsf{Pr}[A'(f(V)) = T_{k^{-1}}^{-1}(f^{-1}(f(V)))] < \frac{1}{P(n)}.$$
$T_{k^{-1}}^{-1}(f^{-1}(f(V))) = T_{k^{-1}}^{-1}(V)$. Let random variable $W$ denote $A'(f(V))$, then by Latin Square's property,
$$\forall x \in \mathbb{Z}_r^n, \mathsf{Pr}[V = x] = \mathsf{Pr}[T_{k^{-1}}^{-1}(V) = x] = \frac{1}{2^n},$$
thus
$$\begin{aligned} \mathsf{Pr}[W = V] &= \sum_{x,y} \mathsf{Pr}[W = x] \cdot \mathsf{Pr}[V = y] \cdot \chi(x = y) \\ &= \sum_{x,y} \mathsf{Pr}[W = x] \cdot \mathsf{Pr}[T_{k^{-1}}^{-1}(V) = y] \cdot \chi(x = y) \\ &= \mathsf{Pr}[W = T_{k^{-1}}^{-1}(V)] < \frac{1}{P(n)} \end{aligned}$$
3) Substitute the notations $V$ and $W$ by their origins.
$$\mathsf{Pr}[A'(f \circ T_k(U_n)) = (f \circ T_k)^{-1}(f \circ T_k(U_n))] < \frac{1}{P(n)}.$$

The proof for $T_k \circ f$ is similar:

1) $A' \circ T_k$ is a polynomial-time algorithm. Thus by one-way function's definition, for every such a probabilistic polynomial-time algorithm $A' \circ T_k$, every positive polynomial $P(\cdot)$, and all sufficiently large $n$
$$\mathsf{Pr}[A'(T_k(f(U_n))) = f^{-1}(f(U_n))] < \frac{1}{P(n)}.$$
2) Based on step 1,
$$\begin{aligned} &\mathsf{Pr}[A'(T_k(f(U_n))) = f^{-1}(f(U_n))] \ = \\ &\quad \mathsf{Pr}[A'(T_k(f(U_n))) = f^{-1}(T_{k^{-1}}^{-1}(T_k(f(U_n))))] < \frac{1}{P(n)}. \end{aligned}$$

3) Therefore,

$$\mathsf{Pr}[A'(T_k \circ f(U_n)) = (T_k \circ f)^{-1}(T_k \circ f(U_n))] < \frac{1}{P(n)}.$$

The cases for $f \circ T_m, T_m \circ f$ as well as for the symmetric cases in $T^{-1}$ $\{f \circ T_{k^{-1}}^{-1}, T_{k^{-1}}^{-1} \circ f, f \circ T_e^{-1}, T_e^{-1} \circ f\}$ can be proved similarly. ∎

Theorem 18 means that function composition on a LaSO does not change the one-way property of any strong one-way function. This stable property implies that function composition $\circ$ can be integrated with nominal construction $\bigstar$ to realize a stable structure. That is, if a LaSO also implements one-way function, then (1) Shamir [19] proved that the values returned by compositions of such one-way functions are indeed unpredictable; (2) More formally, Yao [21, p.88] defined *stable one-way function* with invariance properties and also showed that any stable one-way function $f$ can be used to construct a cryptographically strong pseudorandom number generator.

*Definition 19:* **(One-way Latin Square Oracle):** For a Latin Square Oracle $OWL$ that produces ciphertext $e = OWL(m, k)$ from plaintext $m$ and key $k$, it is a *one-way Latin Square Oracle (OWLaSO)* if its curried result $OWL_m$ is a one-way function. In other words, it is hard to obtain key $k$ from a pair of known plaintext and ciphertext $(m, e)$. □

Due to Kerckhoffs' desiderata, the key possesses all secrecy in the system, thus cryptanalysts have no chance to gain information of $m$ by obtaining $L_k^T$ from the key. As a result, for the counterpart function $L^{T'}(k, m) = L^T(m, k)$, the curried result $L_k^{T'}$ is assumed to be safe. But for the curried result $L_m^T$, it should be a one-way function.

Here we use a commonly used one-way function based on DLP to demonstrate the existence of OWLaSO upon the existence of OWF.

*Definition 20:* Let $p$ be a strong prime in the form of $2 \cdot p' + 1$ where $p'$ is a large prime. Let $g$ be a generator of the multiplicative group $\mathbb{Z}_p^*$. The one-way function "*exponentiation modulo* $p$" $f : \mathbb{Z}_p^* \mapsto \mathbb{Z}_p^*$ is defined as $f(x) = g^x \bmod p$. □

Let $g$ be a generator of the multiplicative group $\mathbb{Z}_p^*$. By the function $m \cdot g^k \bmod p$ used in El Gamal encryption[5] [9], the OWLaSO is comprised of permutations constructed from different plaintexts and different keys raised to the generator (using $\mathbb{Z}_5^*$ as an example):

$$\begin{array}{cc} & \begin{array}{cccccc} & m = & 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} k = \\ \\ \\ \\ \end{array} & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \left[\begin{array}{cccc} e = & 1 \cdot g^1 & 2 \cdot g^1 & 3 \cdot g^1 & 4 \cdot g^1 \\ & 1 \cdot g^2 & 2 \cdot g^2 & 3 \cdot g^2 & 4 \cdot g^2 \\ & 1 \cdot g^3 & 2 \cdot g^3 & 3 \cdot g^3 & 4 \cdot g^3 \\ & 1 & 2 & 3 & 4 \end{array}\right] \end{array}.$$

Elements at the same column are distinct due to the different powers raised to the generator. Any collision contradicts the assumption that $g$ is a generator. Elements at the same row are distinct because of the different plaintexts. Any collision contradicts the uniqueness of multiplicative inverse. Thus the result is a Latin Square (The last row is always the plaintext itself as $g^{p-1} \equiv 1 \bmod p$). In addition, key is not revealed given a known plaintext-ciphertext pair due to the one-way property of DLP.

Not all one-way functions can realize a OWLaSO. Many Feistel structures and S-P Networks used in commercial software are not collision-free. If different keys map same plaintext into same ciphertext, then the cipher is obviously not a Latin Square. RSA function and Rabin function operate over multiplicative group $\mathbb{Z}_{p \cdot q}^*$ where $p$ and $q$ are large primes. The square matrices corresponding to such encryptions are also not Latin Square (due to collisions at the rows and columns corresponding to $p$ or $q$'s multiples). However, for these two one-way functions, the "imperfectness" is measurable so that we can predict how "bad" it is when nominal construction is applied on such non-Latin square matrices. They are "good" enough approximation of OWLaSO when the measured imperfectness is negligible for sufficiently large $n$.

### C. Double one-way Latin Square Oracle (DOWLaSO)

In Definition 19, the curried result $\text{OWLaSO}_k$ is not necessarily a one-way function due to Kerchhoffs' desiderata. If we ignore Kerchhoffs' desiderata and improve the OWLaSO design so that $\text{OWLaSO}_k$ is also a one-way function,

[5]If we define encryption as $(m + 1) \cdot g^{k+1} \bmod p$, then schemes based on $\mathbb{Z}_p^*$ can be used to encrypt messages from $\mathbb{Z}_{p-1}$.

then we obtain *double one-way Latin Square Oracle (DOWLaSO)*.

*Definition 21:* **(Double one-way Latin Square Oracle):** For a Latin Square Oracle $OWL$ that produces ciphertext $e = OWL(m, k)$ from plaintext $m$ and key $k$, it is a *double one-way Latin Square Oracle (DOWLaSO)* if its curried results $OWL_m$ and $OWL_k$ are one-way functions. In other words, it is hard to obtain key $k$ from a pair of known plaintext and ciphertext $(m, e)$, or to obtain plaintext $m$ from a pair of known key and ciphertext $(k, e)$. □

In a DOWLaSO, knowing the cipher key cannot decrypt the ciphertext into plaintext. Thus DOWLaSO cannot be *directly* used in symmetric key encryption schemes. It can be directly applied in other scenarios. One example is from zero-knowledge protocols [4]. For instance, "$g^{m+k}$ mod $p$" is a good candidate for constructing a DOWLaSO. For a strong prime $p$ and generator $g$ selected in multiplicative group $\mathbb{Z}_p^*$, "$g^m \cdot g^k = g^{m+k}$ mod $p$" forms a Latin Square. It is hard to obtain $k$ from any known pair $(m, e)$, or to obtain $m$ from any known pair $(k, e)$. Thus the constructed Latin Square is a DOWLaSO if DLP realizes a "good" one-way function.

Another example is the one-way function "$(g^x)^y = g^{x \cdot y}$ mod $p$" used in Diffie-Hellman key exchange protocol. For a strong prime $p$ and generator $g$ selected in multiplicative group $\mathbb{Z}_p^*$, if the plaintext is not $p-1$, that is, $\forall m \in (\mathbb{Z}_p^* - \{p-1\})$, then $g' = (g^m \bmod p)$ is yet another generator of $\mathbb{Z}_p^*$. Thus "$g^{m \cdot k}$ mod $p$" forms a Latin Square over the set $(\mathbb{Z}_p^* - \{p-1\})$. The Latin Square can be used to do key exchange over the set $\mathbb{Z}_{p-2}$.

Encryption modes of operation is also a potential application for DOWLaSO. In stream cipher modes of operation, pseudorandom keystream is needed in encryption and decryption. In Section VII we will show that flawed NIST's modes of operation can be fixed by substituting OWLaSO with DOWLaSO. This repair is necessary due to the fact that Feistel Structures and S-P Networks are not "good" implementation of DOWLaSO. Given a known pair $(k, e)$, it is always easy to obtain the corresponding plaintext $m$.

### D. Depiction of LaSO, OWLaSO, and DOWLaSO

Later in this paper we will use the following depictions to denote LaSO, OWLaSO, and DOWLaSO in modes of operation design, where they are considered ideal random oracles that can be replaced by "good" enough implementations like Feistel structures and S-P networks.

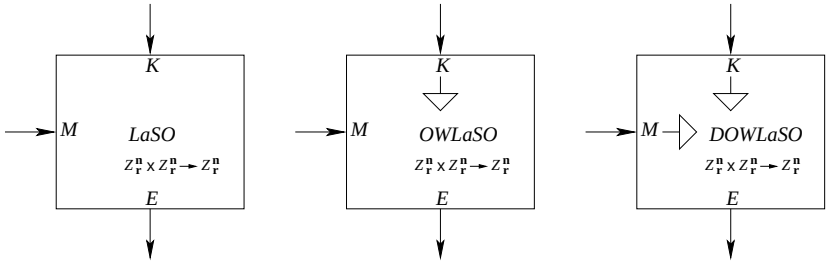


Fig. 2. **Depiction of LaSO, OWLaSO, and DOWLaSO over** $\mathbb{Z}_r^n$

## VI. Pseudorandom Generator

This section discusses how to implement the pseudorandom generator (PRG) depicted in Figure 1. More specifically, we discuss how to realize cryptographically strong pseudorandom generator (CSPRG) on top of Latin Square ciphers.

### A. LaSO as hard-core function

For a one way function $f$, it is unjustified that $f(x)$ will not leak any information about any bit in $x$. However, some specific bits in $x$ or in some efficient function $h(x)$ may remain hidden, even if $f(x)$ and $h$ are given. Such a bit/function is considered a hard-core for $f$, and can be used to construct pseudorandom generators. For example, in a binary system, suppose $f$ is a one-way function and $H$ is a set of boolean functions such that for $h \in H$, $h(x)$ is unpredictable given $f(x), h$. Intuitively, by the following procedure we can produce a random sequence

$h(x_0), h(x_1), \ldots, h(x_m)$ that is unpredictable in polynomial time. (1) Choose a random $h$, a random $x_0$ and output $h(x_0)$; (2) Update $x_{i+1} = f(x_i)$ and output $h(x_i)$ and repeat this step for $i = 0, 1, \ldots, m$; (3) Publish $h$.

For a single bit, the hard-core $b$ is called "hard-core predicate", which is discovered by Blum and Micali [3]. Such $b(x)$ cannot be efficiently predicted given only $f(x)$, thus can be used to construct pseudorandom bit generators. In particular, if predicate $b(x, r)$ is defined as the inner product mod 2 of the binary vector $x$ and $r$, then the predicate $b$ is a hard-core of any one-way function $f$.

*Definition 22:* **(Hard-core predicate):** A polynomial-time-computable predicate $B : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \mapsto \mathbb{Z}_2$ is called a *hard-core* of a one-way function $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^n$ if for every probabilistic polynomial-time algorithm $A'$, every positive polynomial $P(\cdot)$, and all sufficiently large $n$'s,

$$\mathsf{Pr}[A'(f(X_n), R_n) = B(R_n, f(X_n))] < \frac{1}{2} + \frac{1}{P(n)}$$

where $X_n$ and $R_n$ are two independent and uniformly chosen random variables over $\mathbb{Z}_2^n$.

Goldreich and Levin [12] proved that $B$ is a hard-core predicate of any one-way function if $B$ is defined as inner product mod 2. Hard-core functions are similarly defined and discovered. Here we applied a restriction of endomorphism (i.e., $f$ is length-preserving on finite integer groups), since our cryptanalysis on encryption modes of operation is performed on endomorphic one-way functions.

*Definition 23:* **(Hard-core function):** Let $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^n$ be an endomorphic one-way function. Let $H : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^m$ be a polynomial-time-computable function, where $m \leq n$ and each curried $h \in H$ is also a polynomial-time-computable function $h : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^m$. An $\varepsilon(n)$-*oracle* for $H$ is a probabilistic polynomial-time algorithm $A'$ such that

$$\mathsf{Pr}[A'(f(X_n), R_n) = H(R_n, f(X_n))] \geq 2^{-m} + \varepsilon(n)$$

where $X_n, R_n$ are two independent and uniform distributions over $\mathbb{Z}_2^n$. $H$ is called a *hard-core function* for $f$ if no $\varepsilon(n)$-oracle exists for non-negligible $\varepsilon(n)$. When $m = 1$, the hard-core function is called *hard-core predicate*. □

In our cryptanalysis on encryption modes of operation design, we are interested in the case when $m = n$.

*Theorem 24:* A LaSO $T$ over $\mathbb{Z}_r^n$ is a hard-core function of any one-way endomorphic function $f : \mathbb{Z}_r^n \mapsto \mathbb{Z}_r^n$.

***Proof***: At first we can always do radix conversion and convert a LaSO $T$ over $\mathbb{Z}_r^n$ to another equivalent LaSO $T'$ over $\mathbb{Z}_2^{n'}$. This can be done by extending the corresponding Latin Square to $2^{n'}$ wide, where $n' = n\lceil \log_2 r \rceil$. A LaSO $T'$ over $\mathbb{Z}_2^{n'}$ is a sampleable collection of $2^{n'}$ functions. We need to prove the case

$$\mathsf{Pr}[A'(f(X_{n'}), R_{n'}) = T'(R_{n'}, f(X_{n'}))] \ < \ 2^{-n'} + \frac{1}{P(n')}$$

for every probabilistic polynomial-time algorithm $A'$, every positive polynomial $P(\cdot)$, and all sufficiently large $n'$'s.

Because the key follows the uniform distribution $R_{n'}$, the LaSO constitutes a perfect system. Thus $T'(R_{n'}, f(X_{n'}))$ also follows uniform distribution due to perfect system's second property (Definition 1). The output of $T'(R_{n'}, f(X_{n'}))$ can only be predicted with $\frac{1}{2^{n'}} < \frac{1}{2^{n'}} + \frac{1}{P(n')}$ probability. The polynomial relation on length is not affected if we change $n'$ to be $n$ by going back to the $r$-ary system. ∎

*Example 3:* Näslund[13] shows that all bits in "$a \cdot x + b \bmod p$" is hard when $p$ is a prime and $a, b \in \mathbb{Z}_p$. More formally, let one way function $f$ be length-preserving and $n = |x| = |f(x)|$ be the security parameter of $f$, let $\mathbb{P}_k$ denote the set of primes of length $n/k$ ($k < n$). The set of functions

$$H_2^k = \{h(x) = a \cdot x + b \bmod p \mid p \in_U \mathbb{P}_k, a \in_U \mathbb{Z}_p, b \in_U \mathbb{Z}_p\}$$

is a hard-core function for any one-way function. Here the function $h \in_U H_2^k$.

For each specific $p \in \mathbb{P}_k$, $h(x) = a \cdot x + b \bmod p$ constitutes the following Latin Square (the "mod $p$" part is omitted in the matrix for ease of presentation):

$$\begin{array}{cccccccc} & & a & = & 0 & 1 & \cdots & p-1 \\ b= & 0 & \left[\begin{array}{cccccc} h(x) & = & 0 & x & \cdots & (p-1)\cdot x \\ & & 1 & x+1 & \cdots & (p-1)\cdot x+1 \\ & & \vdots & \vdots & \ddots & \vdots \\ & & p-1 & x+(p-1) & \cdots & (p-1)\cdot x+(p-1) \end{array}\right] \\ & 1 & & & & & & \\ & \vdots & & & & & & \\ & p-1 & & & & & & \end{array} \quad (1)$$

It is clear that elements per column are distinct. When $x \bmod p \neq 0$, $x$ has a unique multiplicative inverse, thus elements per row are also distinct.

A straight-forward view of Näslund's result is to treat the input sequence $\{x_i\}$ as a huge number with hybrid radixes. Let a sequence of uniformly selected $p_1, p_2, \ldots, p_{m'} \in_U \mathbb{P}_k$ denote the sequence of primes applied on the input $\alpha = x_1, x_2, \ldots, x_m$ (i.e., $p_i$ is used in $h$ for $x_i$, and truly random $p_i$ is padded if $m' < m$). In Näslund's work $\alpha \in_U \mathbb{Z}_2^{n \cdot m}$. Using a method similar to Theorem 17, we firstly convert the representation of $x$ to hybrid radixes: in a little-endian system we treat the input $x$ as the integer $\alpha = x_m \cdot 2^{(m-1)\cdot n} + \ldots + x_2 \cdot 2^{1 \cdot n} + x_1$, then we change the radixes. $x$ is represented as $\alpha' = x'_{m'} \cdot p_{m'}^{m'-1} + \ldots + x'_2 \cdot p_2^1 + x'_1$. This procedure can be accomplished in polynomial time by padding some truly random bits to $\alpha$ and by repetitively producing the remainder and quotient of $\alpha / p_i$ for all $1 \leq i \leq m'$. Now we have $x'_i \in_U \mathbb{Z}_{p_i}$, thus $h(x'_i) \in_U \mathbb{Z}_{p_i}$ due to perfect system's property (i.e., in Latin Square (1), when $h(\cdot)$ is always a Latin Square over $\mathbb{Z}_{p_i}$ and $a, b \in_U \mathbb{Z}_{p_i}$, the result $h(\cdot) \in_U \mathbb{Z}_{p_i}$). In other words, the output is a sequence of truly random $p_i$-ary integers.

However, Näslund's process is slightly different from the above one. Instead of splitting the entire input into flexible "blocks" in the form of $p_i$-ary integers, the process is applied on fixed $n$-bit "blocks". That is, each $x_i$ is an $n$-bit integer. Nevertheless, this change does not invalidate the hard-core conclusion.

- Case I: For any $x_i \bmod p_i \neq 0$ (i.e., $x_i$ has a multiplicative inverse in $\mathbb{Z}_{p_i}$), nothing changes (note "mod $p_i$" is omitted in Latin Square (1) for ease of presentation).
- Case II: Otherwise (i.e., $x_i \bmod p_i = 0$), it is easy to verify that the probability of selecting such $x_i$ is only $\frac{k}{2^n} < \frac{n}{2^n}$, where $n = |x|$ is the security parameter of the one-way function $f$. This is also a negligible quantity for sufficiently large $n$'s. □

We can use the above analysis to prove Näslund's Theorem in terms of perfect system.

*Theorem 25:* **(Näslund's Theorem)**: For a (strong) one-way function $f$, every bit generated from $H_2^k$ is hard.

***Proof***: Näslund has already proven this theorem in [13]. The sketch previously described in Case I and II will obtain the same result. In the proof we only use the fact that the set $H_2^k$ always returns Latin Square in Case I, and the probability that $H_2^k$ fails to return Latin Square is negligible (Case II).

- For Case I, no matter what $x_i$ is, as long as $x_i \bmod p_i \neq 0$, we always have a perfect system constituted by a Latin Square and its uniformly distributed inputs $a, b \in_U \mathbb{Z}_{p_i}$. Each output $h(x_i)$ is a truly random $p_i$-ary integer. Thus for every probabilistic polynomial-time algorithm $A'$, every positive polynomial $P'(\cdot)$, and all sufficiently large $n' > \lceil \log_2 p_i \rceil$,
$$\mathsf{Pr}[A'(f(X_{n'}), R_{n'}) = h(R_{n'}, f(X_{n'}))] \ < \ 2^{-n'} + \frac{1}{P(n')}.$$
- For Case II, in the worst case, let's assume that a polynomial time algorithm $A'$ can always differentiate truly random $p_i$-ary integers from the ones returned by $h(x)$. This event happens with probability $< \frac{n'}{2^{n'}}$ for the uniform distribution $X_{n'}$.

Combining these two quantities, we have

$$\mathsf{Pr}[A'(f(X_{n'}), R_{n'}) = h(R_{n'}, f(X_{n'}))] \ < \ 2^{-n'} + \frac{1}{P(n')} + \frac{n'}{2^{n'}} = 2^{-n'} + \frac{1}{P'(n')}.$$

Due to L'Hospital's rule, the quantity $\frac{n'}{2^{n'}}$ is less than $\frac{1}{P''(n)}$ for any polynomial $P''(\cdot)$ and all sufficiently large $n$. Then we can find a polynomial $P'$ such that $\frac{1}{P(n')} + \frac{1}{P''(n')} = \frac{1}{P'(n')}$. ∎

*Theorem 26:* **(Generalized Näslund's Theorem)**: Näslund's Theorem is true if we use another collection of Latin Squares to replace the collection of Latin Squares corresponding to "$a{\cdot}x + b \bmod p$" (i.e., collection of Latin Square (1) actualized on argument $x$).

***Proof***: The proof is based on the previous one since we don't care about other mathematical properties of "$a{\cdot}x + b \bmod p$" as long as it constitutes a collection of Latin Squares. The theorem generalizes Näslund's result. For example, there is no need to add the condition $p{\in_U}\mathbb{P}_k$, a prime constant $p{\in}\mathbb{P}_k$ will not affect the validity of the theorem. In addition, either $a$, or $b$, but not both, can be a fixed constant $\in_U \mathbb{Z}_p$. ∎

### *B. Cryptographically strong pseudorandom generator*

Cryptographically strong pseudorandom generators can be constructed upon hard-cores of a one-way function $f$. The following construction is based on Blum-Micali pseudorandom generator [3], except that the hard-core predicate is substituted with a hard-core function.

*Definition 27:* **(Blum-Micali pseudorandom generator):** Let $f$ be an endormorphic one-way function $f : \mathbb{Z}_2^n{\times}\mathbb{Z}_2^n \mapsto \mathbb{Z}_2^n$. Let $b : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^m$ be a polynomial-time-computable hard-core function of $f$, and let $P(\cdot)$ be an arbitrary polynomial satisfying $P(n) > n$. Given a truly random inputs $s, x, y{\in_U}\mathbb{Z}_2^n$, the pseudorandom generator $G$ is defined as $G(s) = \sigma_1\sigma_1 \cdots \sigma_{P(n)}$, where $s_0 \overset{\text{def}}{=} s$, and for every $1{\leq}i{\leq}P(n)$ it holds that $\sigma_i = b(x, s_{i-1})$ and $s_i = f(y, s_{i-1})$. That is, the algorithm $G$ proceeds as follows:

1) Uniformly choose $s_0{\in_U}\mathbb{Z}_2^n$.
2) For $i = 1$ to $P(n)$ do $\sigma_i{\leftarrow}b(x, s_{i-1})$ and $s_i{\leftarrow}f(y, s_{i-1})$, where $x, y{\in_U}\mathbb{Z}_2^n$.
3) Output $\sigma_1\sigma_1 \cdots \sigma_{P(n)}$.

$G$ is a cryptographically strong pseudorandom generator. □

The essential structure of Blum-Micali pseudorandom generator is depicted in Figure 3. In this paper we conjecture that a valid encryption mode of operation design must be a cryptographically strong pseudorandom generator (CSPRG), and in particular such CSPRG can be reduced to Blum-Micali pseudorandom generator.

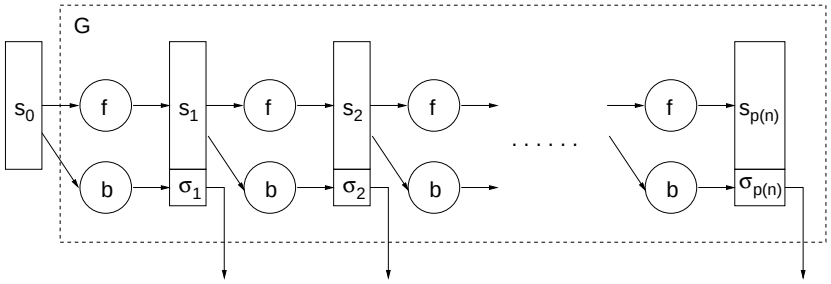


Fig. 3. **Blum-Micali pseudorandom generator** (A binary cryptographically strong pseudorandom generator)

*Proposition 28:* **(Design thesis of encryption modes of operation)**: An encryption mode of operation design should be proven to be equivalent to Blum-Micali pseudorandom generator. □

### *C. Cryptographically strong pseudoperfect system*

By the help of ideal oracles like LaSO, OWLaSO, and DOWLaSO, we can construct cryptographically strong pseudoperfect systems that are equivalent to Blum-Micali pseudorandom generators. Figure 4 and 5 show two equivalent cryptographically strong pseudoperfect systems with key, plaintext, ciphertext depicted in details (they are equivalent due to the symmetric nature of DOWLaSO). In the construction DOWLaSO is used in places of one-way function $f$, and LaSO is used in places of hard-core function $b$. In both figures, $y, s, x$ is instantiated on random seed key $k$, random seed vector $v$, and random plaintext $m$, respectively.

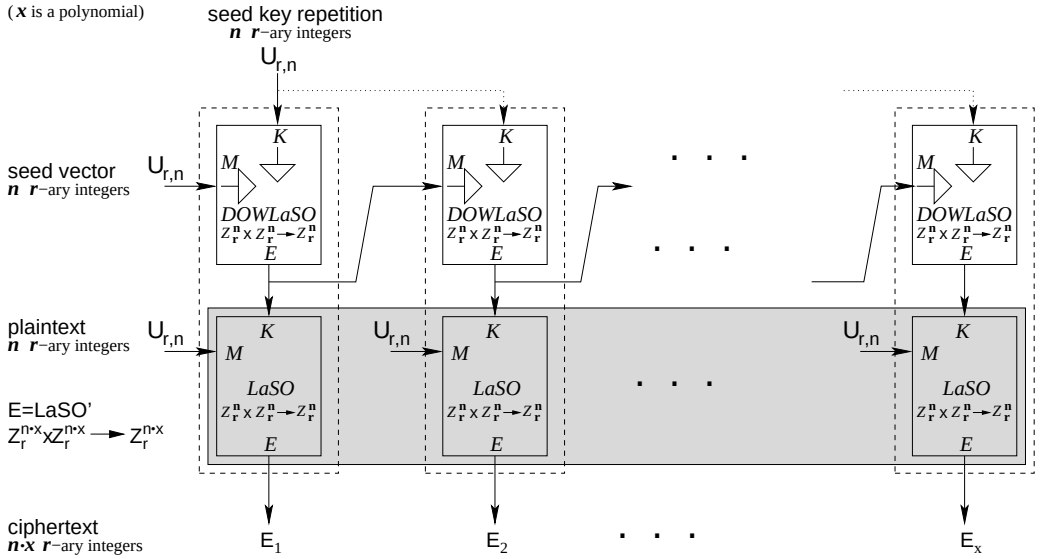


Fig. 4. **Using LaSO and DOWLaSO to implement Blum-Micali pseudorandom generator** (An $r^n$-ary cryptographically strong pseudoperfect system)

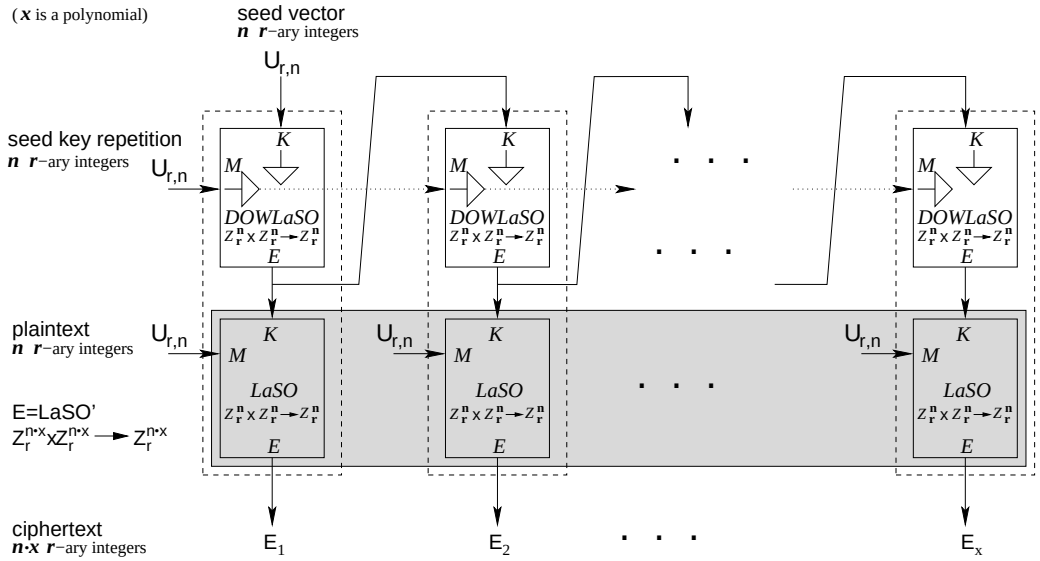


Fig. 5. **An equivalence of Figure 4**

If DOWLaSO is unavailable, for instance, when Feistel structures and S-P networks are used as one-way functions, OWLaSO must be used to construct cryptographically strong pseudoperfect systems. Figure 6 shows the OWLaSO-based cryptographically strong pseudoperfect systems. Figure 7 and 8 demonstrate two possible mistakes in the construction:

1) The only difference between Figure 6 and Figure 7 is the switch between $K$ and $M$ ports at the top-level. As OWLaSO is not a symmetric structure, the one depicted in Figure 7 is *not* a cryptographically strong pseudorandom generator since we cannot find a one-way function composition chain at all.
2) The only difference between Figure 6 and Figure 8 is that OWLaSO at bottom-level is replaced by invertible LaSO. Thus the keystream fed into next function composition could be revealed by a known-plaintext attack. Then the cryptanalyst can reveal the seed key upon the knowledge (i.e., deduce $M$ from $K$ and $E$. The DOWLaSO-based system in Figure 5 is not vulnerable to this attack).

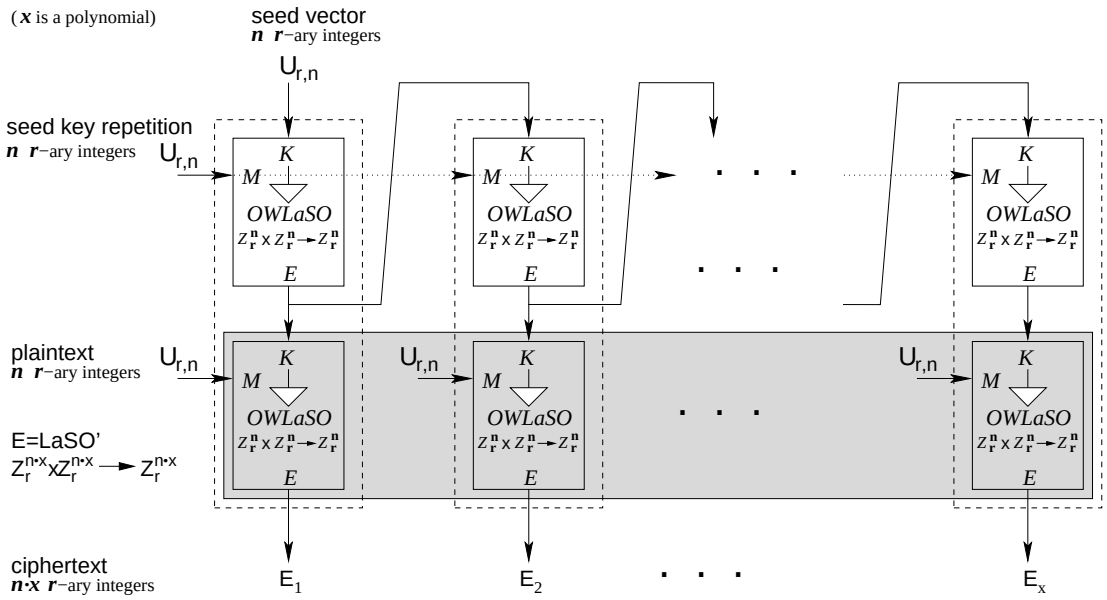


Fig. 6. **Using OWLaSO to implement Blum-Micali pseudorandom generator** (An $r^n$-ary cryptographically strong pseudoperfect system)

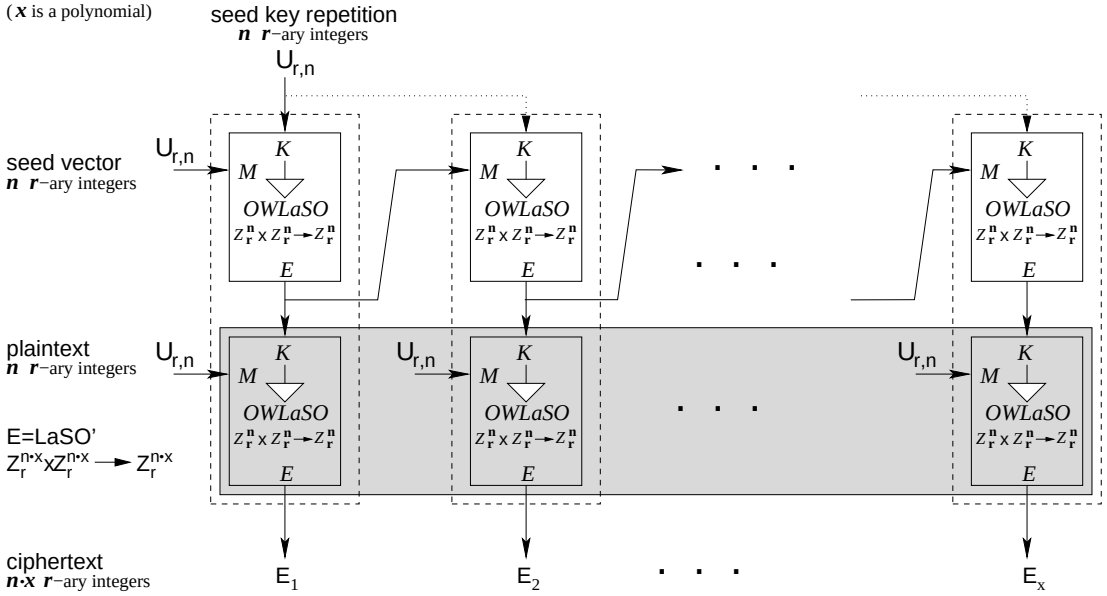


Fig. 7. **Not a valid equivalence of Figure 6**

## VII. Applying the pseudoperfect system model in practice

### A. NIST's standard modes of operation

The pseudoperfect system model proposed in this work can be used to analyze standard encryption modes of operation (OFB,CFB,CTR,CBC) [14], [15] published by National Institute of Standards and Technology (NIST). We will proceed according to the following order: Output-Feedback mode (OFB), Cipher-Feedback mode (CFB), Counter mode (CTR), and Cipher-Block-Chaining mode (CBC).

Cipher FeedBack (CFB) mode, Output Feedback (OFB) mode, and Counter (CTR) mode are stream ciphers based on standard block ciphers. A stream cipher is in general a pseudoperfect system following the diagrams depicted in Figure 4 and Figure 5. The implementation of the invertible LaSO is the bitwise exclusive-OR operation $\oplus$. Note that a cryptographically strong stream cipher is not operating on $L_{2,1}^{N(1)}$. Instead, it operates on $L_{2,n}^{N(1)}$ where $n$ is the output length of the one-way function it employs. The length is determined by cryptographic parameters used

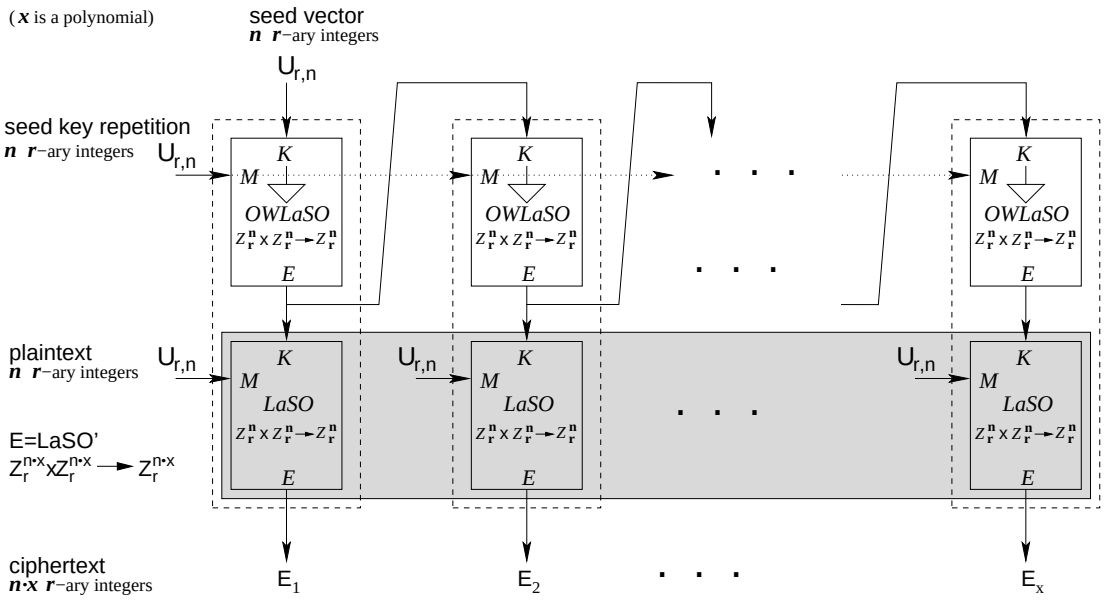


Fig. 8. **Pseudoperfect system not robust against known-plaintext attack**

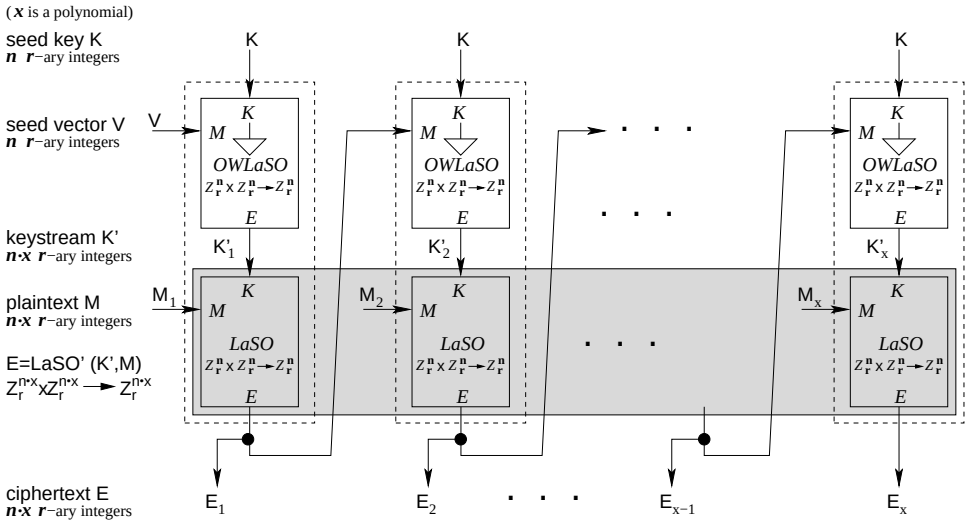


Fig. 9. **Pseudoperfect system corresponding to CFB mode** ($r = 2$, not a valid $2^n$-ary cryptographically strong pseudoperfect system)

in a real implementation, such as the block size in block cipher based stream ciphers, or the register length in shift register based stream ciphers.

Unfortunately, Feistel structures and S-P networks implement OWLaSO rather than DOWLaSO. When DES and AES are used, OFB mode corresponds to the system depicted in Figure 7. It implements a pseudoperfect system, but not a cryptographically strong pseudoperfect system. For CFB mode, it corresponds to the system depicted in Figure 9. Again we cannot find a one-way function composition chain needed in CSPRG. CFB mode is not a cryptographically strong pseudoperfect system.

CTR mode corresponds to the system depicted in Figure 9, where $r = 2$, $n$ is the block cipher's block size, the top-level LaSO is constituted by the Latin Square cipher $e = (k + m + 1) \bmod 2^n$ over the set $\mathbb{Z}_2^n$, the bottom-level LaSO is $L_{2,n}^{N(1)}$, and the mid-level OWLaSO is DES or AES. Again we cannot find a one-way function composition chain needed by Blum-Micali pseudorandom generator. CTR mode is not a cryptographically strong pseudoperfect system.

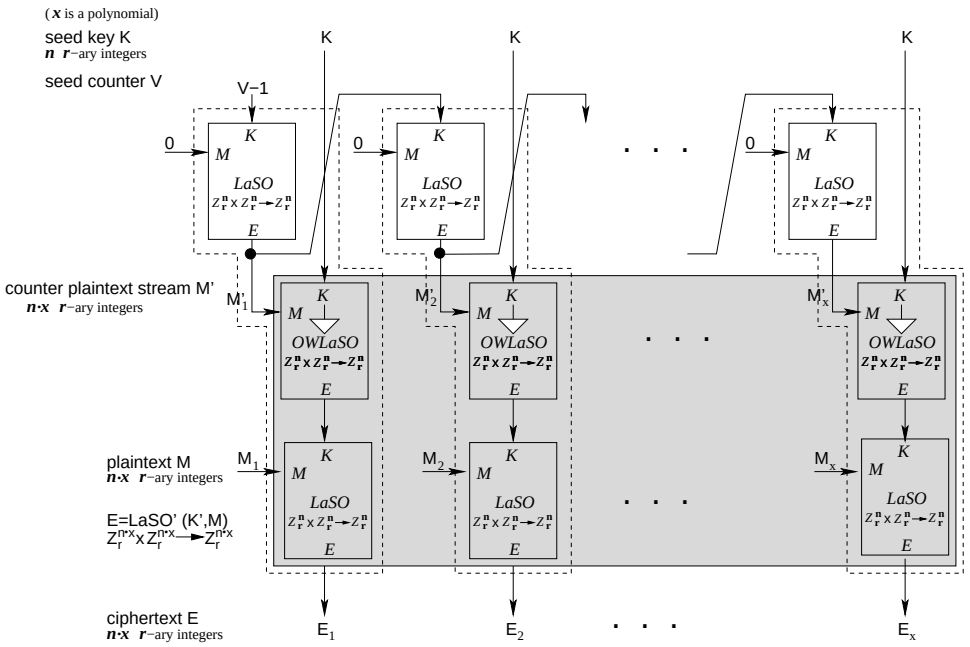


Fig. 10. **Pseudoperfect system corresponding to CTR mode** (Not a valid $r^n$-ary cryptographically strong pseudoperfect system)

$$
\begin{array}{rcl}
 & k = & 0 \quad 1 \quad 2 \quad \cdots \quad 2^n-1 \\
m = & \begin{array}{l} 0 \\ 1 \\ 2 \\ \vdots \\ 2^n-1 \end{array} & \left[ \begin{array}{rcccccc} e = & 1 & 2 & 3 & \cdots & 0 \\ & 2 & 3 & 4 & \cdots & 1 \\ & 3 & 4 & 5 & \cdots & 2 \\ & \vdots & \vdots & \vdots & \ddots & \vdots \\ & 0 & 1 & 2 & \cdots & 2^n-1 \end{array} \right]
\end{array}
$$

CBC mode corresponds to the system depicted in Figure 11. Like the previous stream cipher modes, it is not a cryptographically strong pseudoperfect system due to lack of one-way function composition chain.

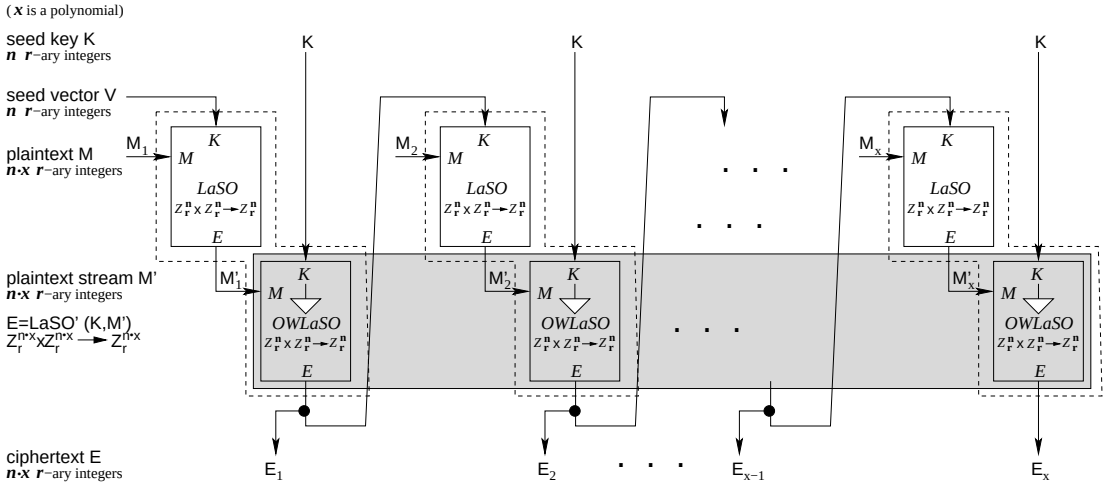


Fig. 11. **Pseudoperfect system corresponding to CBC mode** (Not an $r^n$-ary cryptographically strong pseudorandom generator)

In a nutshell, none of the NIST's standard modes of operation is a cryptographically strong pseudorandom generator or a cryptographically strong pseudoperfect system. They should be replaced by efficient designs that are cryptographically strong pseudoperfect systems.

### B. Exemplary repairs

Efficient repairs for the standard modes of operation are available. Figure 6 is a valid design. Let's denote it by *key-encryption-transform* (KET). Compared to ECB mode, KET encrypts the seed key before an AES encryption.

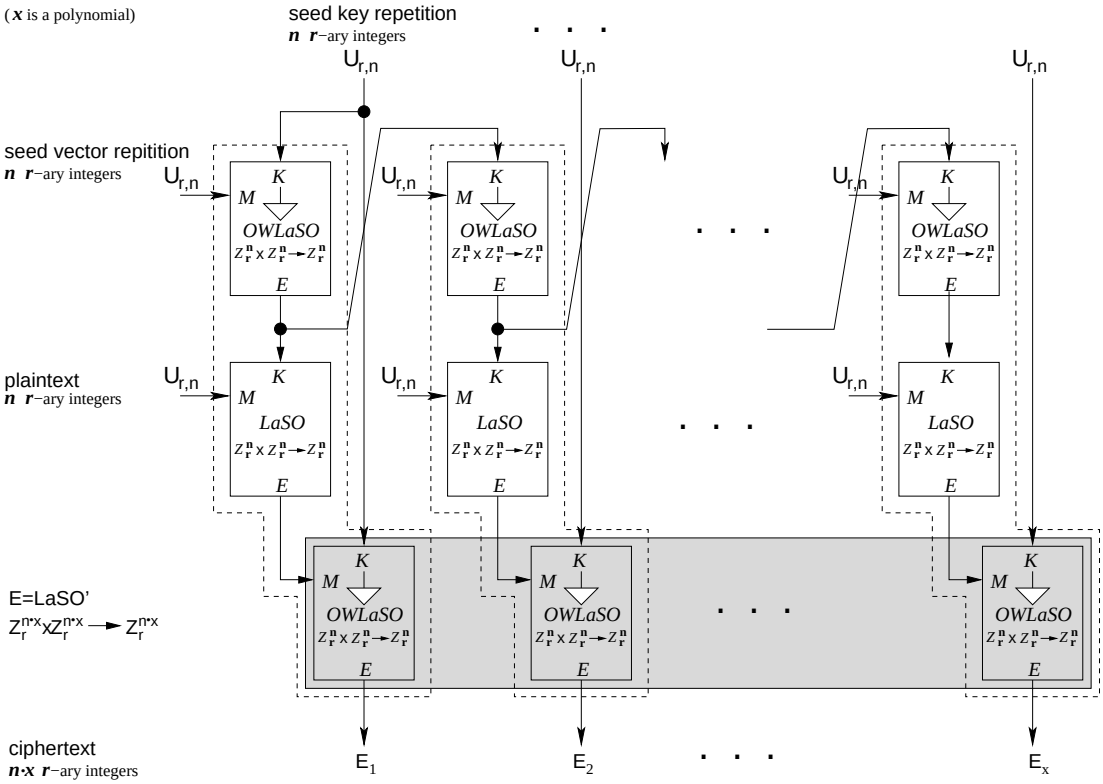


Fig. 12. **Pseudoperfect system corresponding to AONT**

Figure 12 shows Rivest's All-Or-Nothing-Transform (AONT) [17][7]. Intuitively, AONT encrypts the plaintext twice. It is another valid design: (1) The top-level constitutes a one-way function composition chain needed by Blum-Micali CSPRG; (2) Given a shared seed key $K$ and seed vector $V$, the decryptor can always recover the $M$ at bottom-level and $E$ at top-level, then recover the plaintext at the mid-level; (3) The double encryptions at top and bottom levels protect the the $K$ and $M$ used at the mid-level. This avoids the chosen-plaintext discussed in Figure 8; (4) $M$ fed into the bottom-level OWLaSO is already a cryptographically strong pseudorandom ensemble. $\text{OWLaSO}_k$ is a permutation that won't affect the pseudorandomness of this ensemble.

## VIII. Conclusion

This work presents an algebraic model for privacy-oriented cryptographic modes of operation. The proposed model extensively explore various roles of Latin Square cipher in formal cryptanalysis. We show the relation between Latin Square ciphers and following issues: (a) block-by-block encryption modes of operation design, (2) composition with one-way function, (3) hard-core function of one-way function, and finally (4) how to construct cryptographically strong pseudorandom generators from Latin Square based random oracles. As a result, Latin Square ciphers with invariance properties can be used to construct a cryptographically strong pseudorandom generator (CSPRG), and thus a cryptographically strong pseudoperfect system that are treated as the ideal case of encryption modes of operation design.

When the random oracles in the ideal case is replaced by "good" implementations like AES, the security of real world modes of operation design can be analyzed and challenged. We find design flaws in NIST's standard modes of operation and show that efficient repairs are easily achievable.

## REFERENCES


[1] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. In *Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 1997.

[2] M. Bellare and P. Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In *1st ACM conference on Computer and Communications Security (CCS)*, pages 62–73, 1993.

[3] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *Society for Industrial and Applied Mathematics (SIAM) Journal on Computing*, 13(4):850–864, 1984.

[4] D. Chaum and T. Pedersen. Wallet Database with Observers. In *CRYPTO*, pages 89–105, 1993.

[5] D. Coppersmith, D. Johnson, and S. Matyas. Triple DES Cipher Block Chaining with Output Feedback Masking. *IBM Research and Development Journal*, 40(2):253–261, 1996.

[6] H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, 1958.

[7] A. Desai. The Security of All-or-Nothing Encryption: Protecting against Exhaustive Key Search. In M. Bellare, editor, *CRYPTO'00, Lecture Notes in Computer Science*, pages 359–375, 2000.

[8] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[9] T. El-Gamal. A Public-key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology–EUROCRYPT*, pages 10–18, 1984.

[10] H. Feistel. Cryptography and Computer Privacy. *Scientific American*, 228(5):15–23, 1973.

[11] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[12] O. Goldreich and L. A. Levin. A Hard-Core Predicate for all One-Way Functions. In *Symposium on the Theory of Computation (STOC)*, pages 25–32, 1989.

[13] M. Näslund. All Bits in $ax + b$ mod $p$ are Hard. In N. Koblitz, editor, *CRYPTO'96, Lecture Notes in Computer Science 1109*, pages 144–128, 1996.

[14] National Institute of Standards and Technology. FIPS PUB 81: DES Modes of Operation. `http://www.itl.nist.gov/fipspubs/fip81.htm`, 1980.

[15] National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation. `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`, December 2001.

[16] M. O. Rabin. Digital Signatures and Public Key Functions as Intractable as Factorization. Technical Report TM-212, Laboratory of Computer Science, Massachusett Institute of Technology, 1979.

[17] R. L. Rivest. All-or-Nothing Encryption and the Package Transform. In E. Biham, editor, *FSE'97, Lecture Notes in Computer Science 1267*, pages 210–218, 1997.

[18] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *CACM*, 21(2):120–126, 1978.

[19] A. Shamir. On the Generation of Cryptographically Strong Pseudo-Random Sequences. In S. Even and O. Kariv, editors, *International Colloquium on Automata, Languages and Programming (ICALP'81), Lecture Notes in Computer Science 115*, pages 544–550, 1981.

[20] C. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[21] A. C.-C. Yao. Theory and Applications of Trapdoor Functions (Extended Abstract). In *Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.